DOT NET MLG

Compilers: the hype of the 50's returns

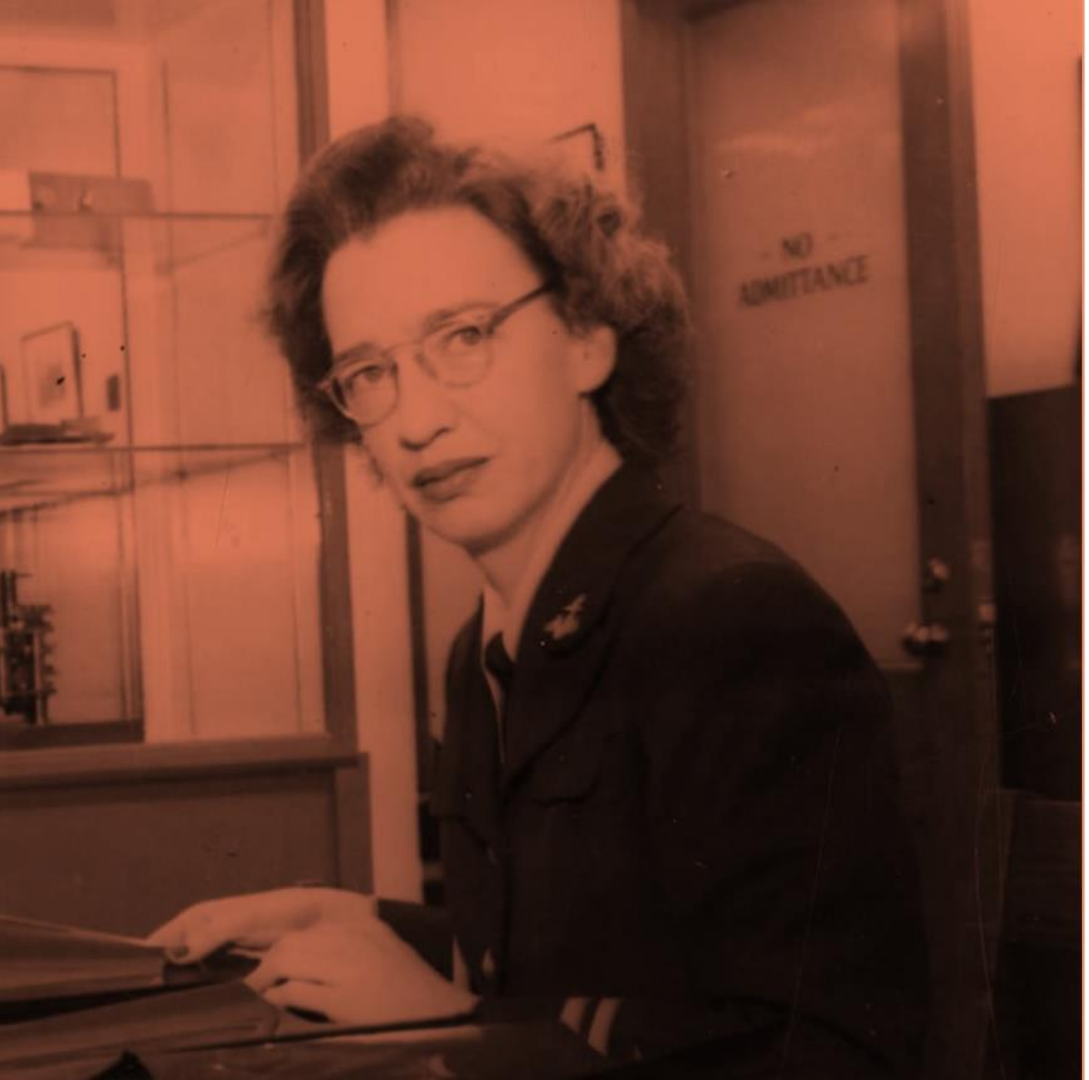
Sequel

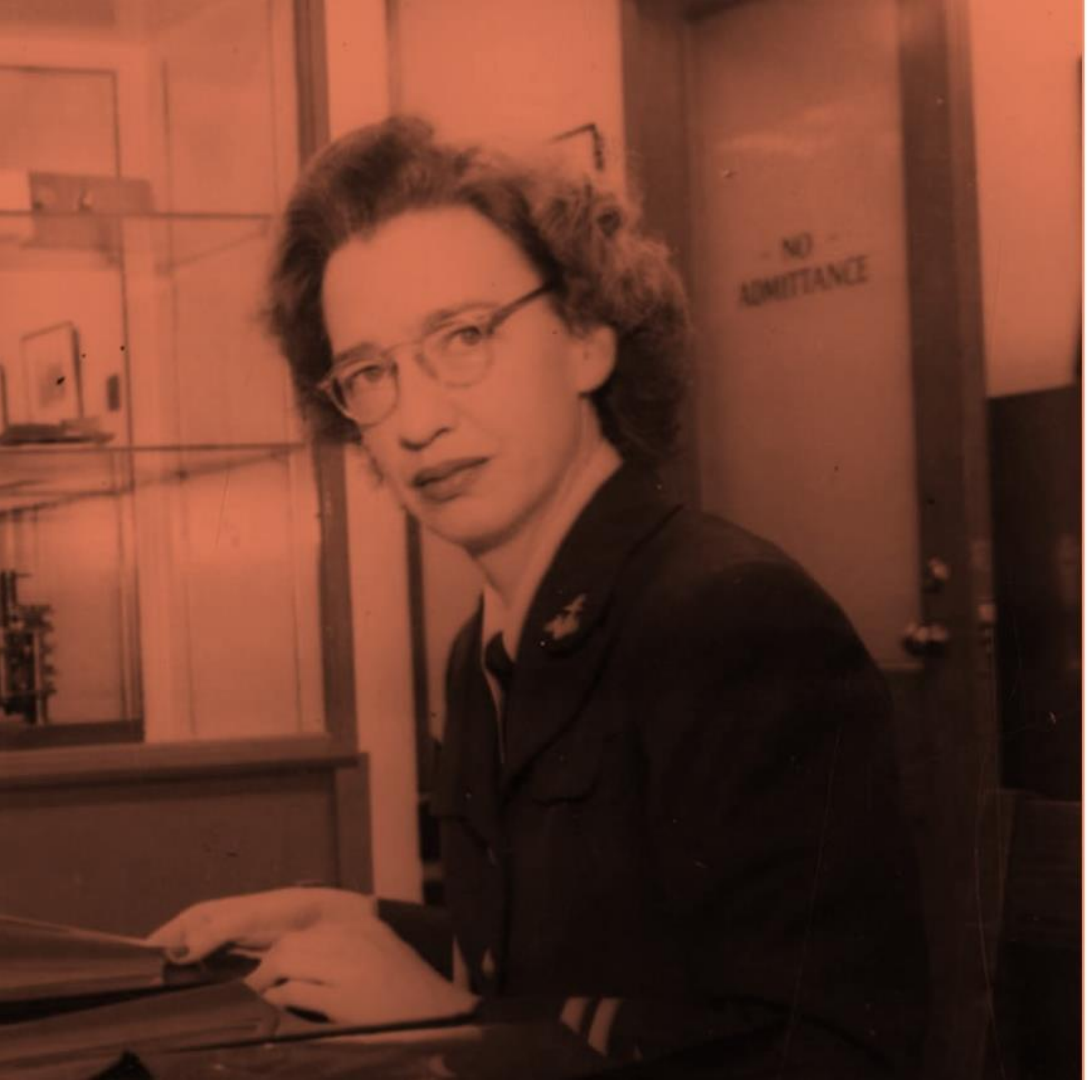
plain concepts

AgioGlobal technology

epam

Bravent IT consulting company


NO
ADMITTANCE



Grace Hopper

Científica de la computación y militar
Estados Unidos, 1906 - 1992

Grace Hopper es considerada la madre de la programación informática y creó el Lenguaje Común Orientado a Negocios (COBOL, por sus siglas en inglés): el primer lenguaje complejo de ordenador. Esta estadounidense obtuvo un doctorado en Matemáticas en Yale en 1934 y fue militar en la Armada estadounidense, donde alcanzó el grado de contraalmirante. Cuando Estados Unidos entró en la Segunda Guerra Mundial, abandonó su trabajo de profesora de matemáticas e ingresó en la Marina.

La Armada la envió a la Universidad de Harvard, donde trabajó como programadora del primer ordenador de gran capacidad, el *Mark I*. Cuando lo vio, pensó: "Caray, es el aparato más bonito que jamás he visto". Tras la guerra, realizó el primer compilador para procesamiento de datos que usaba órdenes en inglés; sin saberlo, Hopper estaba abriendo el camino para hacer más fácil la codificación. En 1986 se retiró de la armada de EE UU de manera definitiva, siendo en ese momento la oficial de más edad. Tras retirarse, continuó dando conferencias, ejerció de consultora y participó en programas educativos.

"Si es una buena idea, continuad y llevadla a cabo. Es mucho más fácil pedir disculpas que conseguir el permiso necesario".

Hi! 👋

I’m Marco, from Plain Concepts

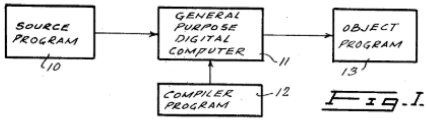

SOURCE PROGRAM
10
GENERAL PURPOSE DIGITAL COMPUTER
11
OBJECT PROGRAM
13
COMPILER PROGRAM
12

Fig. 1.

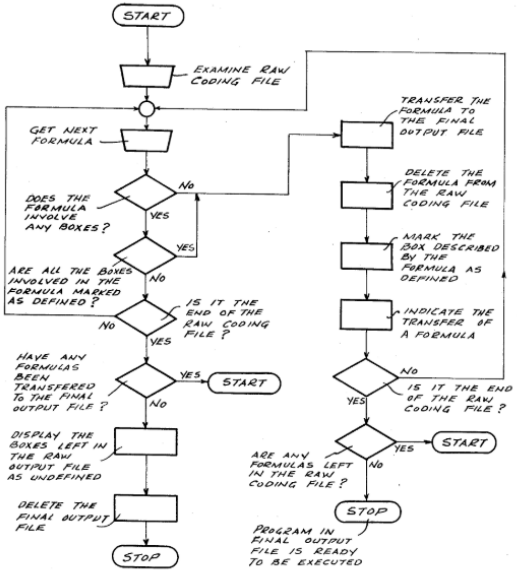
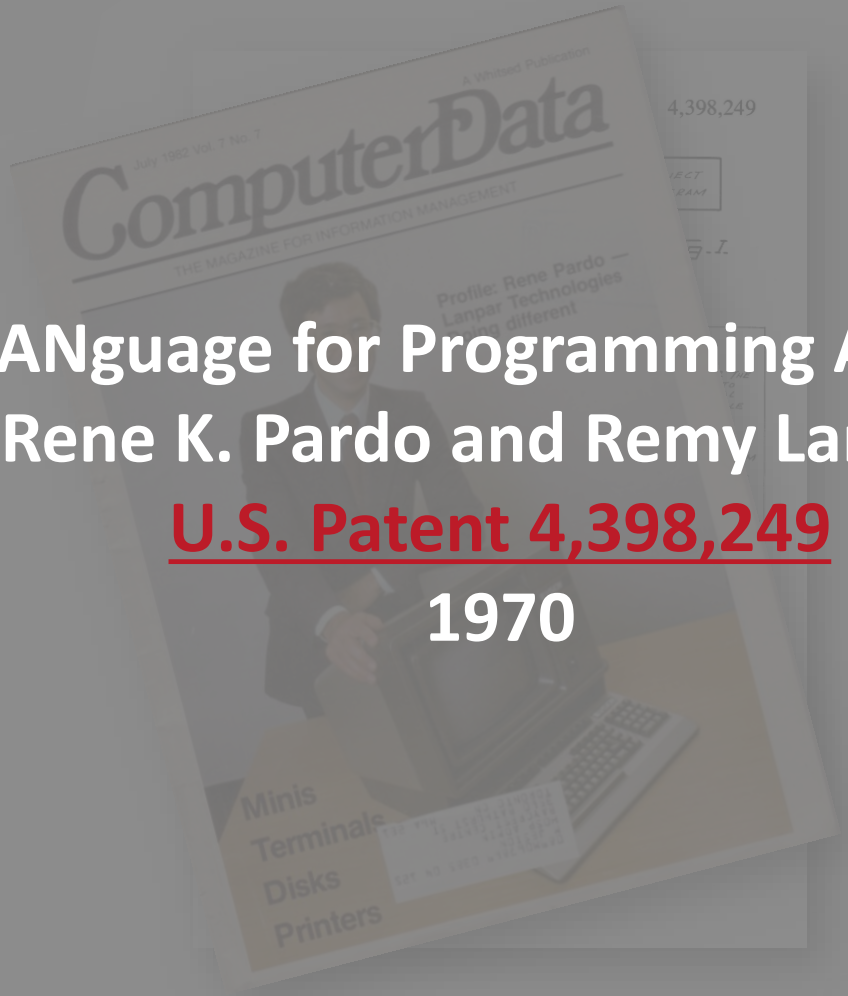

START
EXAMINE RAW CODING FILE
GET NEXT FORMULA
DOES THE FORMULA INVOLVE ANY BOXES?
No
YES
ARE ALL THE BOXES INVOLVED IN THE FORMULA MARKED AS DEFINED?
YES
No
IS IT THE END OF THE RAW CODING FILE?
No
YES
HAVE ANY FORMULAS BEEN TRANSFERED TO THE FINAL OUTPUT FILE?
YES
START
No
DISPLAY THE BOXES LEFT IN THE RAW OUTPUT FILE AS UNDEFINED
DELETE THE FINAL OUTPUT FILE
STOP
TRANSFER THE FORMULA TO THE FINAL OUTPUT FILE
DELETE THE FORMULA FROM THE RAW CODING FILE
MARK THE BOX DESCRIBED BY THE FORMULA AS DEFINED
INDICATE THE TRANSFER OF A FORMULA
No
IS IT THE END OF THE RAW CODING FILE?
YES
ARE ANY FORMULAS LEFT IN THE RAW CODING FILE?
YES
START
No
STOP
PROGRAM IN FINAL OUTPUT FILE IS READY TO BE EXECUTED

Fig. 2.


A Whitsed Publication
July 1982 Vol. 7 No. 7
ComputerData
THE MAGAZINE FOR INFORMATION MANAGEMENT
Profile: Rene Pardo —
Lanpar Technologies
Doing different
Minis
Terminals
Disks
Printers

DOT NET MLG

Sequel



LANPAR — LANguage for Programming Arrays at Random

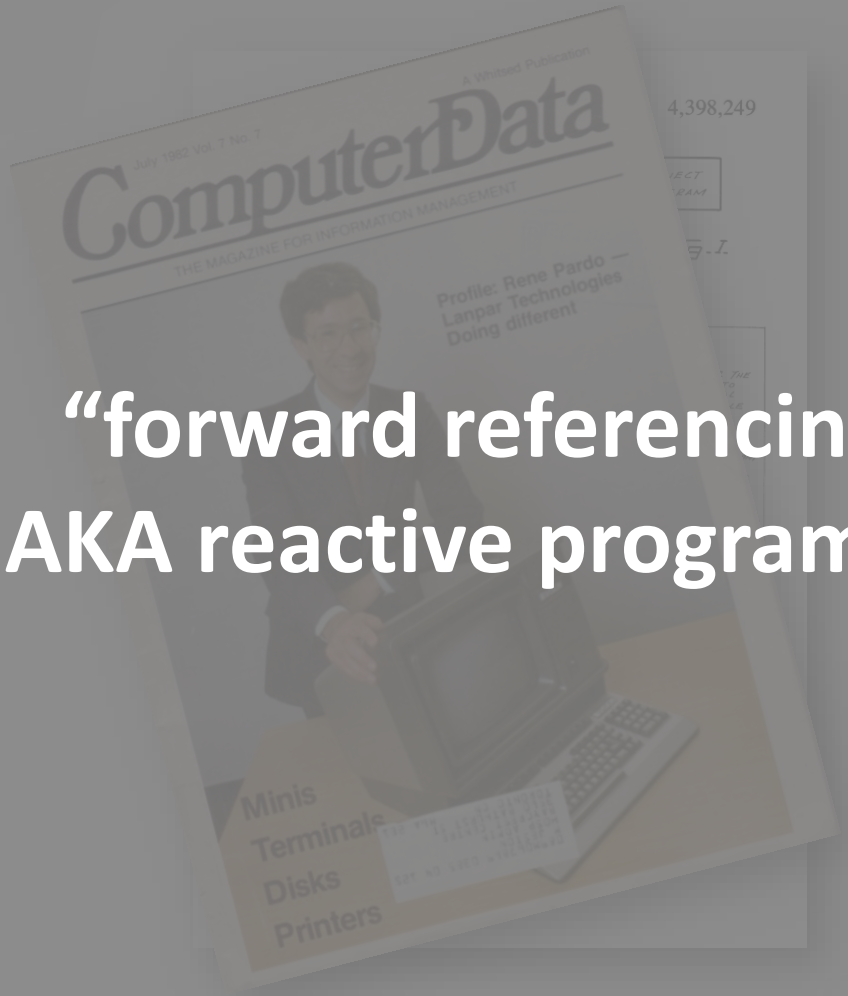
Rene K. Pardo and Remy Landau

U.S. Patent 4,398,249

1970

DOT NET MLG

Sequel



“forward referencing”
AKA reactive programing

A dependency graph…

	A	B	C	D
1	item	price	quantity	total
2	apples	0.69	4	2.76
3	bananas	0.39	6	2.34
4	cantaloupes	2.49	1	2.49
5				7.59

show inputs and outputs

A reconciliation story...

```
import React, { useState } from 'react';
import ReactDOM from 'react-dom';

function App() {
  const [count, setCount] = useState(0);
  const [name, setName] = useState('world');

  function handleClick() {
    setCount(count + 1);
  }

  function handleInput(event) {
    setName(event.target.value);
  }

  return (
    <div className="app">
      <h1>Hello {name}!</h1>
      <input value={name} onInput={handleInput}/>

      <button onClick={handleClick}>
        Clicks: {count}
      </button>
    </div>
  );
}

ReactDOM.render(<App/>, document.querySelector('main'));
```

Hello world!

world | Clicks: 0

```
element div
  className app
  children
    element h1
      children
        text Hello world!
    element input
      value world
    element button
      text Clicks: 0
```

```
element div
  className app
  children
    element h1
      children
        text Hello world!
    element input
      value world
    element button
      text Clicks: 0
```

https://es.reactjs.org/docs/reconciliation.html

A subliminal message...

Optimization	shouldComponentUpdate React.PureComponent useMemo useCallback
Amortization	Concurrent Mode

A hidden engine...

https://tomdale.net/2017/09/compilers-are-the-new-frameworks/

https://github.com/getify/You-Dont-Know-JS

Introducing...

Introducing...

Introducing...

adjective **/svelt/** attractively thin, graceful and stylish

SVELTE

```
<button on:click={handleClick}>
    Clicks: {count}
</button>
```

```
if (changed.count) {
    text.data = `Clicks: ${current.count}`;
}
```

Something has changed...

React

```
state = { count: 0 };

// later...
const { count } = this.state;
this.setState({
    count: count + 1
});

/* or... */

const [count, setCount] = useState(0);

// later...
setCount(count + 1);
```

Something has changed...

Svelte

```
<script>
    let name = 'world';
    let count = 0;
</script>

<h1>Hello {name}</h1>
<input bind:value={name}>

<button on:click={() => count += 1}>
    Clicks: {count}
</button>
```

https://svelte.dev/repl/c5c0f9abe8354a1a9eb398e6ff914040?version=3.12.1

Importing components...

Client side

```
import App from './App.svelte';

new App({
    target: document.querySelector('main'),
    // ... other options
});
```

Server side

```
import App from './App.svelte';

const { html, css } = App.render({ ... });
```

Reactive? 🤔

React

```
import React, { useState } from 'react';

export default function TodoList() {
    const [todos, setTodos] = useState([
        { done: false, text: 'eat' },
        { done: false, text: 'sleep' },
        { done: false, text: 'code' },
        { done: false, text: 'repeat' }
    ]);

    function toggleDone(t) {
        setTodos(todos.map(todo => {
            if (todo === t) return { done: !t.done, text: t.text };
            return todo;
        }));
    }

    const [hideDone, setHideDone] = useState(false);

    function toggleHideDone() {
        setHideDone(!hideDone);
    }

    const filtered = hideDone
        ? todos.filter(todo => !todo.done)
        : todos;

    return (
        <div>
            <label>
                <input
                    type="checkbox"
                    checked={hideDone}
                    onChange={toggleHideDone}
                ></input>

                hide done
            </label>

            <ul>
                {filtered.map(todo => (
                    <li onClick={() => toggleDone(todo)}>
                        {todo.done ? '👍' : ''} {todo.text}
                    </li>
                ))
            </ul>
        </div>
    );
}
```

https://codesandbox.io/s/dotnetmalagareact-not-reactive-qpwvu

Reactive? 🤔

Observable · Apr 10, 2018
The magic notebook for visualization.
By **Mike Bostock**

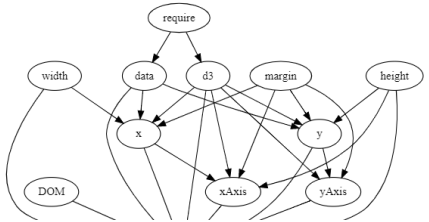
Listed in Introduction 5 forks

How Observable Runs

If you've used other interactive notebooks before, you're probably accustomed to code running from top to bottom. The first cell runs first, the second cell runs second and can reference values defined in the first cell, and so on down the page.

Observable is different: it functions like a spreadsheet, where cells (like formulas) run *automatically* whenever their referenced values change. Observable runs cells independently of their order in the notebook, giving you the flexibility to arrange your cells in whatever order you prefer for literate programming.

Observable uses *topological* order to run cells, as determined by cell references. If cell *B* needs the value of cell *A*, Observable won't run cell *B* until *A* is computed. Likewise whenever the value of cell *A* changes, cell *B* is automatically recomputed. Rather than a linear sequence of cells, an Observable notebook is a directed graph where values flow from top to bottom.



https://observablehq.com/@observablehq/how-observable-runs
@mbostock

Introducing reactivity…

```
let a = 10;

$: b = a + 1;
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/label

Reactive 😄

React

```
import React, { useState } from 'react';

export default function TodoList() {
    const [todos, setTodos] = useState([
        { done: false, text: 'eat' },
        { done: false, text: 'sleep' },
        { done: false, text: 'code' },
        { done: false, text: 'repeat' }
    ]);

    function toggleDone(t) {
        setTodos(todos.map(todo => {
            if (todo === t) return { done: !t.done, text: t.text };
            return todo;
        }));
    }

    const [hideDone, setHideDone] = useState(false);

    function toggleHideDone() {
        setHideDone(!hideDone);
    }

    const filtered = hideDone
        ? todos.filter(todo => !todo.done)
        : todos;

    return (
        <div>
            <label>
                <input
                    type="checkbox"
                    checked={hideDone}
                    onChange={toggleHideDone}
                ></input>

                hide done
            </label>

            <ul>
                {filtered.map(todo => (
                    <li onClick={() => toggleDone(todo)}>
                        {todo.done ? '👍' : ''} {todo.text}
                    </li>
                ))}
            </ul>
        </div>
    );
}
```

Svelte

```
<script>
    let todos = [
        { done: false, text: 'eat' },
        { done: false, text: 'sleep' },
        { done: false, text: 'code' },
        { done: false, text: 'repeat' }
    ];

    function toggleDone(t) {
        todos = todos.map(todo => {
            if (todo === t) return { done: !t.done, text: t.text };
            return todo;
        });
    }

    let hideDone = false;

    const filtered = hideDone
        ? todos.filter(todo => !todo.done)
        : todos;
</script>

<label>
    <input
        type="checkbox"
        bind:checked={hideDone}
    >

    hide done
</label>

<ul>
    {#each filtered as todo}
        <li on:click={() => toggleDone(todo)}>
            {todo.done ? '👍' : ''} {todo.text}
        </li>
    {/each}
</ul>
```

https://svelte.dev/repl/96bfb2fce133460f8eb16cb737def0f4?version=3.12.1

Reactive² 😁😁

Svelte

```
import { readable, derived } from 'svelte/store';

export const time = readable(new Date(), function start(set) {
    const interval = setInterval(() => {
        set(new Date());
    }, 1000);

    return function stop() {
        clearInterval(interval);
    };
});

const start = new Date();

export const elapsed = derived(
    time,
    $time => Math.round(($time - start) / 1000)
);
```

https://svelte.dev/repl/cb9a18ab1a0c45709d33e607da443de0?version=3.12.1

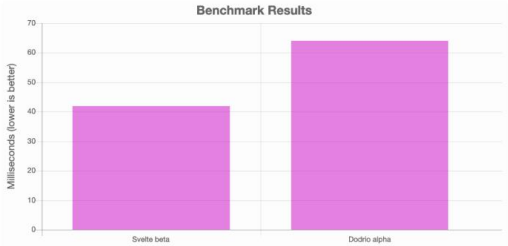
And about performance…

Fast, Bump-Allocated Virtual DOMs with Rust and Wasm

By Nick Fitzgerald

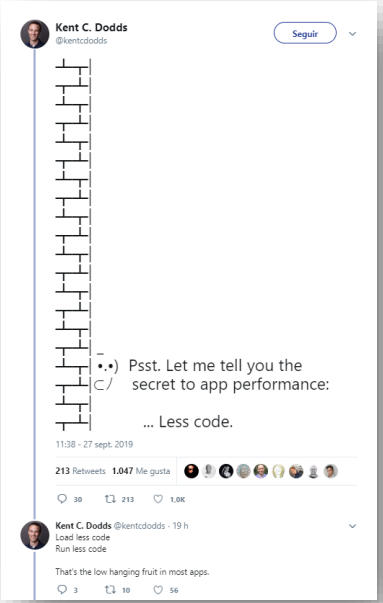
Posted on March 14, 2019 in Featured Article, Rust, and WebAssembly ❤ Share This

Dodrio is a virtual DOM library written in Rust and WebAssembly. It takes advantage of both Wasm's linear memory and Rust's low-level control by designing virtual DOM rendering around bump allocation. Preliminary benchmark results suggest it has best-in-class performance.



https://hacks.mozilla.org/2019/03/fast-bump-allocated-virtual-doms-with-rust-and-wasm/

And about performance…



@kentcdodds

Beyond Svelte…

Sapper

- Next/Gatsby-style framework...
- ...except with much less JavaScript
- Automatic SSR & Code-Splitting

Svelte Native

- Community-driven project
- Based on nativescript-vue

Svelte GL

- Like Three.js, but svelter 😂

Give it a try...

SVELTE

https://svelte.dev/

@Rich_Harris

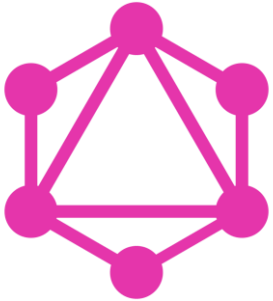
Introducing...

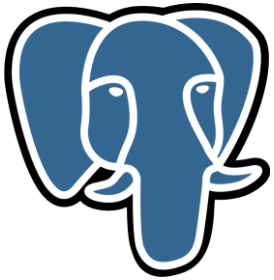
The name Hasura comes from a portmanteau of Asura, the Sanskrit word for demon, and Haskell. Asura refers to daemons, or computer programs that run as background processes. Haskell is the functional programming language we used to build Hasura in, and is Hasura's one true <3.

DOT NET MLG

Sequel

Instant Realtime GraphQL on Postgres





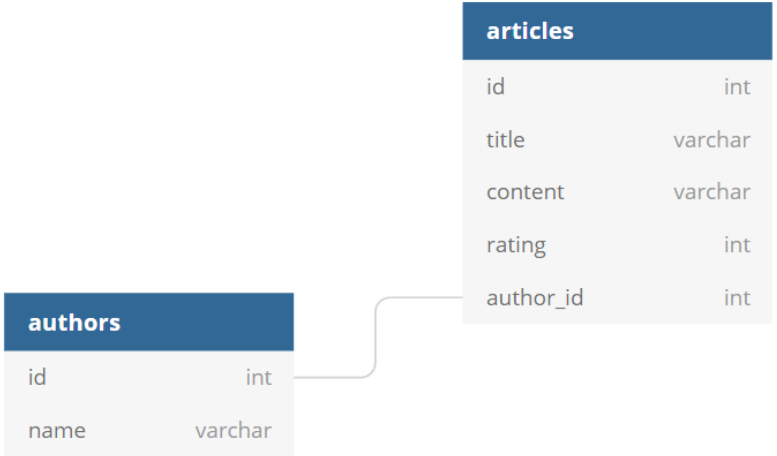
The problem...

There’s data in existing databases

Fetching data is painful 😭

- Building & maintaining APIs
- Security/ access control
- Debugging performance

The model...


articles
id int
title varchar
content varchar
rating int
author_id int
authors
id int
name varchar

The need...

```
{
  author {
    id
    name
    articles {
      id
      title
    }
  }
}
```

The need...

```
{
  author {
    id
    name
    articles {
      id
      title
    }
  }
}
```

Let's use some resolvers...

getAuthor()

getArticle(author_id)

The need...

```
{
  author {
    id
    name
    articles {
      id
      title
    }
  }
}
```

Let's use some resolvers...

getAuthor()

getArticle(author_id)

N+1 Query 😭

The need...

```
{
  author {
    id
    name
    articles {
      id
      title
    }
  }
}
```

Let's use some resolvers...

getAuthor()

getArticle(author_id)

N+1 Query 😭

Use DataLoader instead 👍

https://github.com/graphql/dataloader

Rethinking the solution...

~~Resolve~~ → Compile

GraphQL parser
→ GraphQL AST
→ SQL AST
→ SQL

= 1 query

http://tiny.cc/33eidz

Quick note on ASTs...

https://astexplorer.net/

https://webpack.js.org/api/parser/#program

Rethinking the solution...

~~Resolve~~ → Compile

```
{
  author {
    id
    name
    articles {
      id
      title
    }
  }
}
```

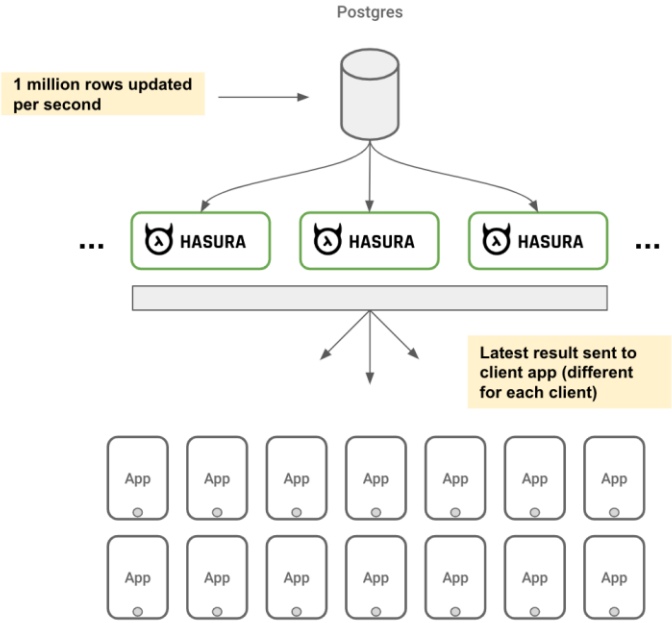
```
SELECT
    author.name
    address.city
    articles.title
    tag.name
FROM
    author, address, articles, tag
WHERE
    author.id = address.author_id,
    author.id = articles.author_id,
    articles.id = tag.article_id
```

And the realtime part?

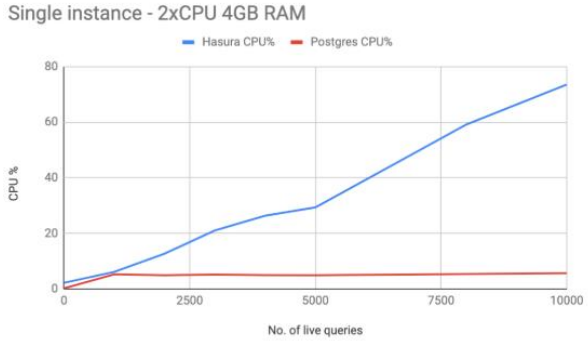
#1: Clients gets events

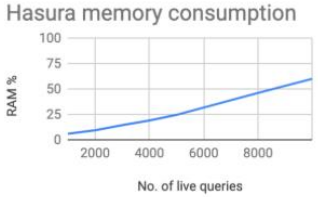
#2: Clients subscribe to queries and get updated results not events

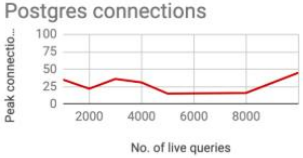
Live queries: data is alive!



Live queries: data is alive!



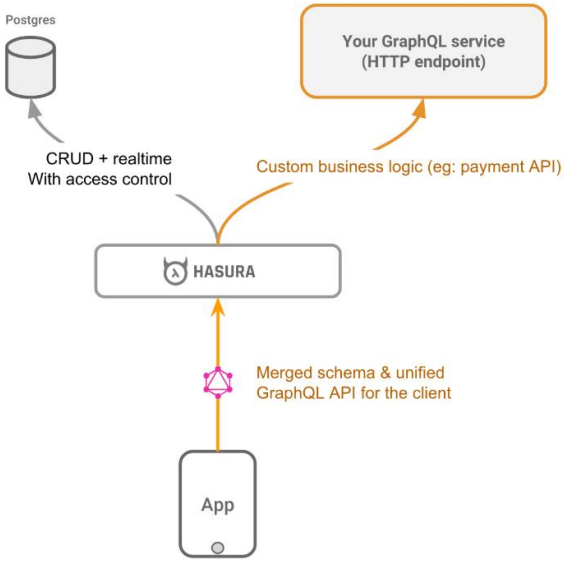




single instance configuration	No. of active live queries	CPU load average
1xCPU, 2GB RAM	5000	60%
2xCPU, 4GB RAM	10000	73%
4xCPU, 8GB RAM	20000	90%

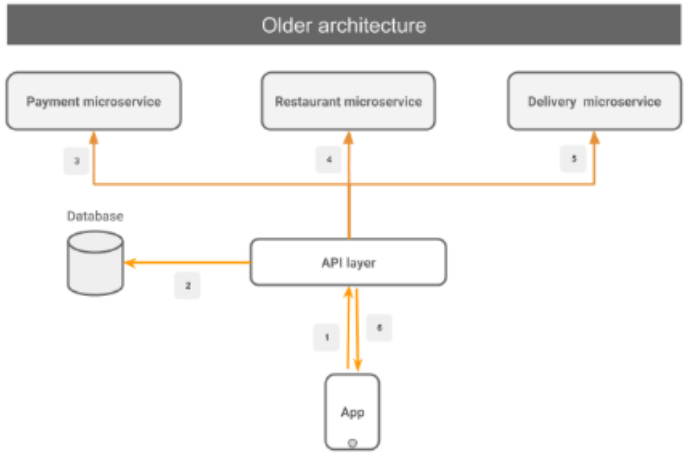
https://github.com/hasura/graphql-engine/blob/master/architecture/live-queries.md

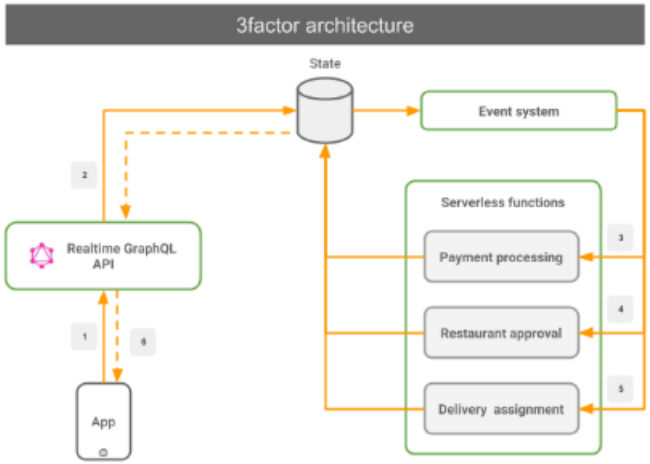
Bringing tables and APIs together



https://docs.hasura.io/1.0/graphql/manual/remote-schemas/

3factor app





https://3factor.app/

3factor app

Factor #1: Realtime GraphQL

Use GraphQL for a very simple and flexible frontend developer workflow.

Low-latency
Support subscriptions

https://3factor.app/

3factor app

Factor #2: Reliable eventing

Remove in-memory state manipulation in your backend APIs and persist them as atomic events instead.

Your event system should have the following 2 properties:

Atomic: Mutations to the application state should atomically create event(s).
Reliable: Events once emitted should be delivered (to any consumer) atleast once.

https://3factor.app/

3factor app

Factor #3: Async serverless

Write business logic as event handlers.

The serverless backends should follow few best-practices:

Idempotent: The code should be prepared for atleast-once (for same event) delivery of events.
Out-of-order: Events may not be guaranteed to be received in the order of creation. The code should not depend on any expected sequence of events.

https://3factor.app/

Give it a try...

https://hasura.io/
@tanmaigo
@rajoshighosh

Show me the code...

...and the slides 😄

https://github.com/beasync/svelte-apollo-app

Thanks! 👋

Sequel

plain concepts

AgioGlobal technology

<epam>

Bravent IT consulting company